МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсивная обработка иерархических списков
Вариант 15

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Фиалковский М. С.

Санкт-Петербург
2019

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, с использованием базовых функций их рекурсивной обработки.

Постановка задачи.

1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2) разработать программу, использующую рекурсию;
3) сопоставить рекурсивное решение с итеративным решением задачи;
4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Вариант 15: проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих подсписках одинаково, при этом атомы могут отличаться);

Описание алгоритма.

Для начала программа должна рекурсивно считать данные, проверить их корректность и занести их в список. Для этого считываем очередной символ строки. Если это атом, добавляем его к иерархическому списку, если это список – рекурсивно заносим его в список.

Далее для определения структурной идентичности двух иерархических списков необходимо, чтобы при их параллельном переборе они одновременно указывали на атом или список. Если один из элементов списка указывает на атом или на список, а другой указывает на следующий парный элемент списка, то тогда списки не идентичны.

```
сравнение_списков (структура_списка1, структура_списка2) {
        если (спиок_1 пустой и список_2 пустой)
            вернуть true
```

```
        еще если (один из списков не пустой)
            вернуть false
        еще если (список_1 атом и список_2 атом)
            вернуть true
        иначе
            вернуть сравнение_следующих_списков() и сравнение_списков()
        }

сравнение_следующих_списков (структура_списка1, структура_списка2) {
        если (спиок_1 пустой и список_2 пустой)
            вернуть true
        еще если (один из списков не пустой)
            вернуть false
        иначе
            вернуть сравнение_следующих_списков() и сравнение_списков()
        }
```

Спецификация программы.

Программа предназначена для проверки идентичности двух иерархических списков.

Программа написана на языке C++ с использованием фреймворка Qt. Входными данными являются символы английского алфавита и скобки - считываются из полей lineEdit или из файла. Выходными данными являются промежуточные значения вычисления выражения и конечный результат. Данные выводятся в qDebug(), результат показывается в всплывающем окне.

Тестирование.

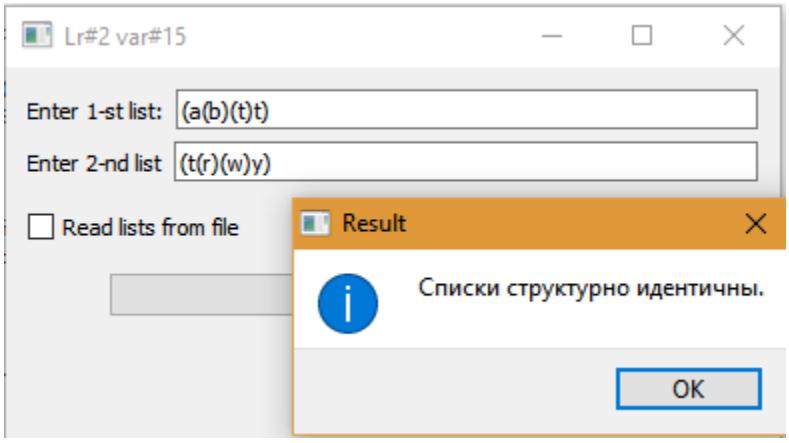


Рисунок 1- сравнение списков

Остальные тесты приведены в табл. 1.

Таблица 1 – Результаты тестирования программы

Ввод	Результат
(qwe()) (abc())	Списки идентичны
() ()	Списки идентичны
(qwer) (qwer)	Списки идентичны
((qwer()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv)) ((qwer()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv))	Списки идентичны
(weq(weq((weq)(weq)(we))egdb)egdb) (weq(weq((weq)(weq)(we))egdb)egdb)	Списки идентичны
(qwe) (qwer)	Разное количество элементов в скобках
(qwe()ty) (qwerty)	Разные элементы в списках
qwe()rty qwe()rty	Строка должна начинаться с открывающей

	скобки
(qwe(rty))uio (qwe(rty))uio	Строка должна заканчиваться закрывающей скобкой
(qwe(qwerq)(qwer) (qwe(qwerq)(qwer)	Несоответствие количества открывающих и закрывающих скобок

Анализ алгоритма.

Алгоритм работает за линейное время от размера списка. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою очередь накладывает ограничение на количество вложенных списков, а также затраты производительности на вызов функций.

Описание функций и СД.

Класс-реализация иерархического списка MyList содержит умный указатель на вложенный список (Head), умный указатель на следующий элемент списка (Tail), флаг, для определения атома, значение атома. Умные указатели были использованы во избежание утечек памяти.

Статический метод класса для проверки входных данных на корректность:

```
static bool checkStr(const std::string& str);
```

Принимает на вход ссылку на строку, проверяет размер и структуру строки, возвращает true, если строка корректна, и false в ином случае.

Статический метод класса для создания иерархического списка:

```
static bool buildList(MyListP& list, const std::string& str);
```

Принимает на вход ссылку на строку и ссылку на умный указатель на список, проверяет корректность строки и вызывает функции рекурсивного создания списка.

Приватный метод класса для считывания вложенных списков:

```
  static void readData(MyListP& list, const char prev,
std::string::const_iterator& it,
        const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если предыдущий элемент не равен “(“, создается атом, в ином случае вызывает считывание следующего списка.

Приватный метод класса для считывания следующих списков:

```
static void readSeq(MyListP& list, std::string::const_iterator& it,
        const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если строка пустая – происходит return. Если элемент равен “)”, создается пустой список. В ином случае рекурсивно считываются вложенные и следующие списки, затем объединяются в текущем списке.

Статический метод класса для сравнения двух списков:

```
static bool compareList(MyListP firstList, MyListP secondList, size_t depth = 0);
```

Принимает на вход два умных указателя на списки, глубину рекурсии для отладки и возвращает true, если списки идентичны, и false в ином случае. Сначала списки проверяются на пустоту, они должны быть одновременно пустыми или нет. Далее списки проверяются на атомы. Если списки не пустые, рекурсивно вызывается проверка для вложенных и следующих списков.

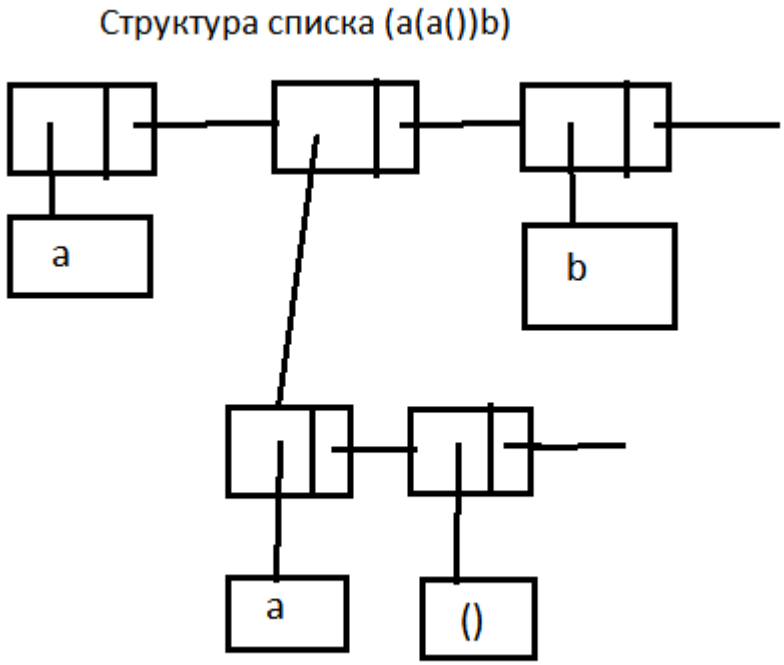
Приватный статический метод класса для сравнения двух следующих списков.

```
static bool compareSeq(MyListP firstList, MyListP secondList, size_t depth = 0);
```

Принимает на вход два умных указателя на списки, глубину рекурсии для отладки и возвращает true, если списки идентичны, и false в ином случае. Прове-

ряется, что они одновременно пустые или нет, далее рекурсивно вызываются функции сравнения.

Рисунок 2 – Структура иерархического списка



Выводы.

В ходе работы я вспомнил работу с классами и контейнерами в C++. Научился решать задачи обработки иерархических списков, с использованием базовых функций их рекурсивной обработки и сравнивать два иерархических списка на структурную идентичность.

Приложение А. Исходный код программы.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include <QMessageBox>
#include <QDebug>
#include <QString>
#include <QDir>
#include <QStringList>
#include <QFileSystemModel>
#include <string>
#include "mylist.h"


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_readFile_checkBox_clicked();

    void on_compare_pushButton_clicked();

    void on_test_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QTextStream* in;
    QFile* file;
    QDir* dir;
};

#endif // MAINWINDOW_H
```

mylist.h

```
#ifndef MYLIST_H
#define MYLIST_H

#include <QObject>
#include <QDebug>
#include <vector>
#include <string>
#include <iostream>
#include <memory>

class MyList : public QObject
{

    /*
     * Класс для работы с иерархическими списками
    */
    Q_OBJECT
public:
    typedef std::shared_ptr<MyList> MyListP;

    explicit MyList(QObject *parent = nullptr);
    ~MyList();

    MyList& operator=(const MyList& list) = delete;
    MyList(const MyList& list) = delete;

    MyListP getHead() const;
```

```
    MyListP getTail() const;
    bool isNull() const;
    bool getIsAtom() const;
    char getAtom() const;

    static bool buildList(MyListP& list, const std::string& str);
    static bool compareList(MyListP firstList, MyListP secondList, size_t depth = 0);
    friend QDebug operator<< (QDebug out, const MyListP list);

private:
    static bool checkStr(const std::string& str);
    static void readData(MyListP& list, const char prev, std::string::const_iterator& it,
                         const std::string::const_iterator& end);
    static void readSeq(MyListP& list, std::string::const_iterator& it,
                        const std::string::const_iterator& end);
    static MyListP cons(MyListP& head, MyListP& tail);
    static bool compareSeq(MyListP firstList, MyListP secondList, size_t depth = 0);
    void createAtom(const char ch);
    void print_seq(QDebug& out) const;

private:
    bool isAtom;
    MyListP head;
    MyListP tail;
    char atom;
};

#endif // MYLIST_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>


//var #15


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow window;
    window.setWindowTitle("Lr#2 var#15");
    window.show();

    return app.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    file = new QFile;
    in = new QTextStream;
    dir = new QDir;
}


MainWindow::~MainWindow()
{
    delete ui;
    delete in;
    delete file;
    delete dir;
}


void MainWindow::on_readFile_checkBox_clicked()
{
    //выбор считывания из файла или из окна
    if (ui->readFile_checkBox->isChecked()) {
```

```
        file->close();

        ui->textLabel_0->setEnabled(false);
        ui->inputFirstList_lineEdit->setEnabled(false);
        ui->textLabel_1->setEnabled(false);
        ui->inputSecondList_lineEdit->setEnabled(false);

        QString fileName = QFileDialog::getOpenFileName(this,
                                                         "Open file");
        file->close();
        file->setFileName(fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        ui->openFile_textLabel->setText(fileName);
    }
    else {
        file->close();

        ui->textLabel_0->setEnabled(true);
        ui->inputFirstList_lineEdit->setEnabled(true);
        ui->textLabel_1->setEnabled(true);
        ui->inputSecondList_lineEdit->setEnabled(true);
        ui->openFile_textLabel->setText("none");
    }
}


void MainWindow::on_compare_pushButton_clicked()
{
    qDebug();
    qDebug() << "Сравнение списков:";

    std::string firstStr = "";
    std::string secondStr = "";

    //считывание строк из файла или из окна
    if (ui->readFile_checkBox->isChecked()) {
        firstStr = in->readLine().toStdString();
        secondStr = in->readLine().toStdString();
    }
    else {
        firstStr = ui->inputFirstList_lineEdit->text().toStdString();
        secondStr = ui->inputSecondList_lineEdit->text().toStdString();
    }

    //создание списков, проверка на корректность
    MyList::MyListP firstList(new MyList);
    MyList::MyListP secondList(new MyList);

    if (MyList::buildList(firstList, firstStr) &&
            MyList::buildList(secondList, secondStr)) {
        qDebug() << "Строки корректны, списки созданы";
    }
    else {
        qDebug() << "Строки некорректны!";
        QMessageBox::critical(this, "Result", "Входные данные неккоректны.");
        return;
    }

    qDebug() << "__Считанные списки__:";
    qDebug() << "Первый список:" << firstList;
    qDebug() << "Второй список:" << secondList;

    //сравнение списков
    if (MyList::compareList(firstList, secondList)) {
        qDebug() << "Списки структурно идентичны.";
        QMessageBox::information(this, "Result", "Списки структурно идентичны.");
    }
    else {
        qDebug() << "Списки НЕ структурно идентичны.";
        QMessageBox::warning(this, "Result", "Списки НЕ структурно идентичны.");
    }

    ui->readFile_checkBox->setChecked(false);
    ui->textLabel_0->setEnabled(true);
    ui->textLabel_1->setEnabled(true);
    ui->inputFirstList_lineEdit->setEnabled(true);
    ui->inputSecondList_lineEdit->setEnabled(true);
```

```
    ui->openFile_textLabel->setText("none");
}


void MainWindow::on_test_pushButton_clicked()
{
    /*
     * Функция тестирования. Тестовые данные считываются из папки Tests
    */

    qDebug() << "Тестирование.";

    std::string firstStr = "";
    std::string secondStr = "";
    dir->cd(QApplication::applicationDirPath() + "/Tests");
    QStringList listFiles = dir->entryList(QStringList("*.txt"), QDir::Files);

    for (auto fileName : listFiles) {
        if (fileName == "." || fileName == "..")
            continue;

        qDebug();
        qDebug() << "Тестовые данные из файла:" << fileName;

        file->close();
        file->setFileName(dir->path() + "/" + fileName);
        file->open(QFile::ReadOnly | QFile::Text);
        in->setDevice(file);

        while (!in->atEnd()) {
            firstStr = in->readLine().toStdString();
            secondStr = in->readLine().toStdString();

            MyList::MyListP firstList(new MyList);
            MyList::MyListP secondList(new MyList);

            if (MyList::buildList(firstList, firstStr) &&
                MyList::buildList(secondList, secondStr)) {
                qDebug() << "Строки корректны, списки созданы";
            }
            else {
                qDebug() << "Строки некорректны!";
                continue;
            }

            qDebug() << "__Считанные списки__:";
            qDebug() << "Первый список:" << firstList;
            qDebug() << "Второй список:" << secondList;

            //сравнение списков
            if (MyList::compareList(firstList, secondList)) {
                qDebug() << "_Списки структурно идентичны._";
            }
            else {
                qDebug() << "_Списки НЕ структурно идентичны._";
            }
            qDebug();
        }
    }
}
```

mylist.cpp

```
#include "mylist.h"

MyList::MyList(QObject *parent) : QObject(parent)
{
    /*
     * По умолчанию объект класса является пустым списком
    */

    isAtom = false;
    atom = 0;
    head = nullptr;
    tail = nullptr;
}


MyList::~MyList()
```

```
{
    /*
     * Т.к в классе используются умные указатели, освбождение
     * памяти происхоит автоматически
    */
}


bool MyList::isNull() const
{
    /*
     * Возвращает true, если элемент является нулевым списком,
     * false - в ином случае
    */

    return (!isAtom && head == nullptr && tail == nullptr);
}


MyList::MyListP MyList::getHead() const
{
    /*
     * Если элемент не атом - возращает указатель на вложенный список,
     * в ином случае - nullptr
    */

    if (!isAtom) {
        return head;
    }
    else {
        std::cerr << "Error: Head(atom)\n";
        return nullptr;
    }
}


MyList::MyListP MyList::getTail() const
{
    /*
     * Если элемент не атом - возращает указатель на следующий список,
     * в ином случае - nullptr
    */

    if (!isAtom) {
        return tail;
    }
    else {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
}


bool MyList::getIsAtom() const
{
    /*
     * Если элемент атом - возращает true,
     * в ином случае - false
    */

    return isAtom;
}


MyList::MyListP MyList::cons(MyListP& head, MyListP& tail)
{
    /*
     * Функция создания списка
    */
    if (tail != nullptr && tail->getIsAtom()) {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
    else {
        MyListP tmp(new MyList);
        tmp->head = head;
        tmp->tail = tail;
        return tmp;
    }
```

```
}


void MyList::print_seq(QDebug& out) const
{
    /*
     * Функция печати Tail
    */

    if (!isNull()) {
        out << this->getHead();

        if (this->getTail() != nullptr)
            this->getTail()->print_seq(out);
    }
}


QDebug operator<<(QDebug out, const MyList::MyListP list)
{
    /*
     * Перегрузка оператора вывода для отладки пограммы
    */

    if (list == nullptr || list->isNull()) {
        out << "()";
    }
    else if (list->getIsAtom()) {
        out << list->getAtom();
    }
    else {
        out << "(";

        out << list->getHead();
        if (list->getTail() != nullptr)
            list->getTail()->print_seq(out);

        out << ")";
    }

    return out;
}


bool MyList::checkStr(const std::string& str)
{
    /*
     * Функция проверки корректности входных данных,
     * принимает на вход ссылку на строку, проверяет размер и структуру строки,
     * возвращает true, если строка корректна, и false в ином случае
    */

    qDebug() << "Проверка на корректность:" << str.c_str();
    int countBracket = 0;

    if (str.size() < 2)
        return false;

    if (str[0] != '(' || str[str.size() - 1] != ')')
        return false;

    size_t i;
    for (i = 0; i < str.size(); ++i) {
        char elem = str[i];
        if (elem == '(')
            ++countBracket;
        else if (elem == ')')
            --countBracket;
        else if (!isalpha(elem))
            return false;

        if (countBracket <= 0 && i != str.size()-1)
            return false;
    }

    if (countBracket > 0 || i != str.size()) {
        return false;
    }
```

```
    qDebug() << "Строка корректна.";
    return true;
}


void MyList::createAtom(const char ch)
{
    /*
     * Создается объект класса - атом
    */
    this->atom = ch;
    this->isAtom = true;
}


void MyList::readData(MyListP &list, const char prev, std::string::const_iterator &it,
                      const std::string::const_iterator& end)
{
    /*
     * Функция считывания данных. Считывает либо атом, лио рекурсивно считывает список
    */

    if (prev != '(') {
        list->createAtom(prev);
    }
    else {
        readSeq(list, it, end);
    }
}


void MyList::readSeq(MyListP& list, std::string::const_iterator&it,
                     const std::string::const_iterator& end)
{
    /*
     * Функция считывания списка. Рекусривно считывает данные и спискок и
     * добавляет их в исходный.
    */

    MyListP headList(new MyList);
    MyListP tailList(new MyList);

    if (it == end)
        return;

    if (*it == ')') {
        ++it;
    }
    else {
        char prev = *it;
        ++it;
        readData(headList, prev, it, end);
        readSeq(tailList, it, end);
        list = cons(headList, tailList);
    }
}


bool MyList::buildList(MyListP& list, const std::string& str)
{
    /*
     * Функция создания иерархического списка. Принимает на вход ссылку
     * на строку, проверяет корректность строки и вызывает приватный метод
     * readData().
    */

    qDebug() << "В список добавляется содержимое следующих скобок:" << str.c_str();

    if (!checkStr(str))
        return false;

    auto it_begin = str.cbegin();
    auto it_end = str.cend();
    char prev = *it_begin;
    ++it_begin;
    readData(list, prev, it_begin, it_end);

    return true;
}
```

```
char MyList::getAtom() const
{
    /*
     * Функция возвращает значение атома
    */
    if (isAtom) {
        return atom;
    }
    else {
        std::cerr << "Error: getAtom(!atom)\n";
        return 0;
    }
}


bool MyList::compareList(MyListP firstList, MyListP secondList, size_t depth)
{
    /*
     * Функция сравнения двух списков.
     * Принимает на вход два списка, глубину рекурсии для отладки и
     * возвращает true, если списки идентичны, и false в ином случае.
     *
     * Сначала списки проверяются на пустоту, они должны быть одновременно
     * пустыми или нет. Далее списки проверяются на атомы. Если списки не пустые, рекурсивно
     * вызывается проверка для вложенных и следующих списков
    */

    std::string dbgStr = "";
    for(size_t i = 0; i < depth; ++i)
        dbgStr += " ";

    qDebug() << dbgStr.c_str() << "Сравниваются списки:" << firstList << "и" << secondList;

    if (firstList == nullptr && secondList == nullptr) {
        qDebug() << dbgStr.c_str() << "Оба списка пустые";
        return true;
    }
    else if ((firstList != nullptr && secondList == nullptr) ||
             (firstList == nullptr && secondList != nullptr)) {
        qDebug() << dbgStr.c_str() << "Один список пустой, другой нет";
        return false;
    }
    else if (firstList->getIsAtom() && secondList->getIsAtom()) {
        qDebug() << dbgStr.c_str() << "Оба списка атомы";
        return true;
    }
    else if (firstList->isNull() && secondList->isNull()) {
        qDebug() << dbgStr.c_str() << "Оба списка пустые";
        return true;
    }
    else if ((firstList->isNull() && !secondList->isNull()) ||
             (!firstList->isNull() && secondList->isNull())) {
        qDebug() << dbgStr.c_str() << "Один список пустой, другой нет.";
        return false;
    }
    else if ((firstList->getIsAtom() && !secondList->getIsAtom()) ||
             (!firstList->getIsAtom() && secondList->getIsAtom())){
        qDebug() << dbgStr.c_str() << "Один список атом, другой нет";
        return false;
    }
    else {
        bool result_compareList = compareList(firstList->getHead(), secondList->getHead(), depth+1);
        bool result_compareSeq = compareSeq(firstList->getTail(), secondList->getTail(), depth);
        return result_compareSeq && result_compareList;
    }
}


bool MyList::compareSeq(MyListP firstList, MyListP secondList, size_t depth) {
    /*
     * Функция сравнения Tail-списков (находящихся на одном уровне вложенности)
     * Проверяется, что они одновременно пустые или нет, далее рекурсивно
     * вызываются функции сравнения
    */

    std::string dbgStr = "";
    for(size_t i = 0; i < depth; ++i)
```

```
        dbgStr += " ";

    qDebug() << dbgStr.c_str() << "Сравниваются списки:" << firstList << "и" << secondList;

    if ((firstList == nullptr && secondList == nullptr)) {
        qDebug() << dbgStr.c_str() << "Оба списка пустые";
        return true;
    }
    else if ((firstList != nullptr && secondList == nullptr) ||
             (firstList == nullptr && secondList != nullptr)) {
        qDebug() << dbgStr.c_str() << "Один список пустой, другой нет";
        return false;
    }
    else if ((firstList->isNull() && secondList->isNull())) {
        qDebug() << dbgStr.c_str() << "Оба списка пустые";
        return true;
    }
    else {
        bool result_compareList = compareList(firstList->getHead(), secondList->getHead(), depth+1);
        bool result_compareSeq = compareSeq(firstList->getTail(), secondList->getTail(), depth);
        return result_compareSeq && result_compareList;
    }
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>387</width>
    <height>185</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Ignored" vsizetype="Ignored">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QLabel" name="textLabel_0">
        <property name="text">
         <string>Enter 1-st list:</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QLineEdit" name="inputFirstList_lineEdit"/>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_3">
      <item>
       <widget class="QLabel" name="textLabel_1">
        <property name="text">
         <string>Enter 2-nd list</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QLineEdit" name="inputSecondList_lineEdit"/>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_5">
```

```
  <item>
   <widget class="QCheckBox" name="readFile_checkBox">
    <property name="text">
     <string>Read lists from file</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_3">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Minimum</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QLabel" name="label">
    <property name="text">
     <string>File path:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="openFile_textLabel">
    <property name="text">
     <string>none</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="compare_pushButton">
    <property name="styleSheet">
     <string notr="true"/>
    </property>
    <property name="text">
     <string>Compare</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
```

```
        </size>
       </property>
      </spacer>
     </item>
    </layout>
   </item>
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout_4">
     <item>
      <spacer name="horizontalSpacer_4">
       <property name="orientation">
        <enum>Qt::Horizontal</enum>
       </property>
       <property name="sizeHint" stdset="0">
        <size>
         <width>40</width>
         <height>20</height>
        </size>
       </property>
      </spacer>
     </item>
     <item>
      <widget class="QPushButton" name="test_pushButton">
       <property name="text">
        <string>Test</string>
       </property>
      </widget>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <widget class="QMenuBar" name="menuBar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>387</width>
    <height>21</height>
   </rect>
  </property>
 </widget>
 <widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>
```